Bounds from Slopes

David M. Gay
Optimization and Uncertainty Quantification
Sandia National Laboratories *
Albuquerque, NM

March 24, 2010

Abstract

Sometimes it is desirable to compute good bounds on the possible values of an algebraic expression involving variables only known to lie in prescribed finite intervals. While interval arithmetic gives rigorous bounds, if some variables appear more than once in the expression, bounds from interval arithmetic can be very pessimistic. Propagation of Taylor series has long been known as a way to obtain much tighter bounds. More recently, researchers have shown that *slope* computations often give tighter bounds than Taylor series. First-order slope computations are fairly straightforward, but second-order slope computations sometimes give still better bounds. This paper reviews interval, Taylor-series and slope computations, provides some proofs, presents a possibly new bound optimization, and discusses implementation approaches based on expression graphs and operator overloading, and describes a still unfinished, open-source implementation for experimenting with expression graphs.

*Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE–AC04–94AL85000. This document is to be released as SAND2010-1794P.

1 Introduction

Mathematical models often take the form of equations or optimization problems whose solutions represent some quantity of interest. Such models commonly involve parameters — input values — that are not known exactly, perhaps because they can only be (or have only been) measured approximately or because they depend on future events or other sources of uncertainty. In some cases, theory or long observation may suggest probability distributions for uncertain parameters. More often, reasonable bounds on such parameters may be known or can be guessed. Sometimes, e.g., in manufacturing, bounds on some input parameters can be reduced if necessary, but such reductions increase costs. For various reasons (e.g., assessing the validity of a model, controlling costs, guarding against catastrophic failure, or performing global optimization), it is useful to compute bounds on computed quantities from bounds on the input parameters involved.

This paper is concerned with the special but often useful case of computing bounds on algebraic expressions that involve input parameters that lie in specified intervals. Crude bounds can be computed with interval arithmetic, and arbitrarily tight bounds can be computed by Taylor-series methods. Of interest in this paper are variants of Taylor-series methods that use *slopes*, which are bounds on divided differences. First- and second-order slope computations often give tighter bounds than do corresponding first- and second-order Taylor-series computations.

Slope computations are described by various papers, some of which are cited below, in the literature on interval analysis. The next section briefly reviews interval analysis and interval arithmetic. Sections 3 and 5 discuss first- and second-order slopes. Component-wise expansions are appealing, as explained in §4. Section 6 discusses some implementation strategies and an ongoing, open-source implementation for experimenting with slope computations on expression graphs. Concluding remarks appear in §7.

2 Interval Analysis and Arithmetic

Interval analysis encompasses techniques for computing bounds on expressions from bounds on the operands.

Interval analysis is a form of real and complex analysis in which some variables are constrained to lie in specified intervals (or disks); a primary goal

is to compute bounds on function values and solution sets. As explained in Rump's nice (draft) survey paper [24], the roots of interval analysis go back at least to the 1930's, but it was Ramon Moore [18, 19, 20] who brought this area to the attention of many researchers, leading to significant progress. (Though retired, Moore is still active [21].)

The earliest and simplest form of interval analysis is interval arithmetic, in which bounds on an expression are computed operation by operation, using only bounds on the operands. Rules for the elementary operations $\{+, -, \times, \div\}$ are straightforward: if $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$ are intervals ($\underline{x} \le \overline{x}$, $\underline{y} \le \overline{y}$), then exact interval arithmetic on them is defined by

$$\begin{aligned}
\mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
\mathbf{x} - \mathbf{y} &= [\underline{x} + \overline{y}, \overline{x} + \underline{y}] \\
\mathbf{x} \times \mathbf{y} &= [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})] \qquad (1) \\
\mathbf{x} \div \mathbf{y} &= [\min(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y}), \max(\underline{x}/\underline{y}, \underline{x}/\overline{y}, \overline{x}/\underline{y}, \overline{x}/\overline{y})], \qquad (2)
\end{aligned}$$

assuming $\underline{y} > 0$ or $\overline{y} < 0$ in (2). Of course, (1) and (2) can be broken into simpler cases involving tests on the signs of the operands. For example, if $\underline{x} \ge 0$ and $\underline{y} > 0$ then $\mathbf{x} \times \mathbf{y} = [\underline{x}\underline{y}, \overline{x}\overline{y}]$ and $\mathbf{x} \div \mathbf{y} = [\underline{x}/\overline{y}, \overline{x}/\underline{y}]$.

In computer implementations, we can use directed roundings [14] to compute *outer approximations* of interval arithmetic.

If each operand occurs only once in an algebraic expression, then evaluating the expression with interval arithmetic delivers optimal bounds on the expression's value. Much more commonly, some variables appear several times, and interval arithmetic may then yield decidedly pessimistic bounds, because each appearance of a variable is in effect treated as though it could have a value separate from its other appearances. For example, if $x \in [-1, 2]$, then $x^2 \in [0, 4]$, but computing $[-1, 2] \times [-1, 2]$ with interval arithmetic gives $[-4, 4]$.

For discussing overestimation, it is useful to define the width of a set $S \subset \mathbb{R}$ by

$$w(S) = \sup_{\zeta, \xi \in S} |\zeta - \xi|. \qquad (3)$$

For an interval $\mathbf{x} = [\underline{x}, \overline{x}]$, this is just $w(\mathbf{x}) = \overline{x} - \underline{x}$. For $f : \mathbb{R}^n \to \mathbb{R}$ and $X \subset \mathbb{R}^n$, let $f(X) = \{f(x) : x \in X\}$ denote the range of f over X. Given an outer approximation $F \supset f(X)$ of this range, let

$$r(F, f(X)) = w(F)/w(f(X)) - 1 \qquad (4)$$

denote the relative excess with of F as an approximation to $f(X)$. Extending the width definition to $X \subset \mathbb{R}^n$ so $w(X) = \max_{i=1}^n \max_{x,y \in X} |e_i^T(x-y)|$ is the maximum width of X in the direction of any standard unit vector e_i, we would like to compute approximations F such that

$$r(F, f(X)) = O(w(X)), \tag{5}$$

i.e., the excess width is no more than a constant times the domain width. That way, by breaking a given domain into small enough subdomains, we can compute an overall approximation with relative excess width no more than a prescribed $\epsilon > 0$. This in general is not possible with simple interval arithmetic.

Taylor-series approximations (which are described, e.g., in [20]) can yield much tighter bounds than simple interval arithmetic. Suppose the input parameters mentioned above are represented as n independent variables x_i $(1 \le i \le n)$ that lie in specified finite intervals: $x_i \in [\underline{x}_i, \overline{x}_i]$, and that we are interested in an algebraic computation of the form

$$x_i = f_i(x_1, ..., x_{i-1}) \tag{6}$$

for $i = n+1, n+2, ..., m$, where f_i is either an elementary arithmetic operation or an elementary function, usually involving only one or two of the arguments x_j $(j < i)$. After choosing a point $(z_1, ..., z_n)$ (such as the midpoints $z_i = \frac{1}{2}(\underline{x}_i + \overline{x}_i)$ of the input intervals) and an expansion order $d \ge 1$, we can recur a Taylor series approximation

$$T_i(x_1, ..., x_n) = \sum_{j=0}^{d-1} \sum_{|\sigma|=j} c_{i,\sigma} \prod_{k \in \sigma} (x_k - z_k) + \sum_{|\sigma|=d} \mathbf{R}_{i,\sigma} \prod_{k=1}^{n} (x_k - z_k)^{\sigma_k} \tag{7}$$

in which $|\sigma| = j$ is an n-tuple of nonnegative integers summing to j, the $c_{i,\sigma}$ are constants (or, for rigor, small intervals computed with interval arithmetic) and the $\mathbf{R}_{i,\sigma}$ are interval bounds on the suitably scaled d-th order partials of f_i over the range of the arguments to f_i. Of course, the complexity of this approach grows as n^d, but the approach could be feasible to use for $d = 1$ or $d = 2$ when n is not too large. (It works well for reasonably large d when $n = 1$, but that is another story.) For a given $f = f_m$ computed by (6) and $X = \prod_{i=1}^n [\underline{x}_i, \overline{x}_i]$ with $w(X)$ not too large, Taylor-series approximations (7) of order d can produce an outer approximation F with $r(F, x(F)) = O(w(X)^d)$; in principle, with $d = 1$ we can already achieve (5). But as the next sections show, more efficient computations are often possible.

3 First-Order Slopes

For $d = 1$, we can use interval bounds on first derivatives for the $R_{i,\sigma}$ in (7). That is, if $X \subset \mathbb{R}^n$ is a Cartesian product of finite intervals and $F_i'(X) \supseteq \frac{\partial f}{\partial x_i}(X)$, then evaluating

$$f(z) + \sum_{i=1}^{n} F_i'(X)(X - z) \tag{8}$$

in interval arithmetic gives a set of values that contains $f(X)$.

Krawczyk and Neumaier [16] (and for $n > 1$, Neumaier [22]) have proposed a choice for the $R_{i,\sigma}$ that often works better than (8): first-order slopes. For the moment, let $n = 1$. Given $f : \mathbb{R} \to \mathbb{R}$ for $x, z \in \mathbb{R}$ with $x \neq z$, the *slope* $f[x, z]$ is defined by

$$f[x, z] = \frac{f(x) - f(z)}{x - z}. \tag{9}$$

When f is continuously differentiable, $\lim_{x \to z} f[x, z] = f'(z)$, in which case we define $f[z, z] \equiv f'(z)$. Then for any $x \in \mathbb{R}$, we have

$$f(x) = f(z) + f[x, z](x - z). \tag{10}$$

For a finite interval $X \subset \mathbb{R}$, by computing an outer approximation $F[X, z]$ to $f[X, z]$ (i.e., a set $F[X, z] \subset \mathbb{R}$ with $\{f(x, z) : x \in X\} \subseteq F[X, z]$), we can obtain an outer approximation of $f(X)$ by evaluating $f(x) + F[X, z](X - z)$ with interval arithmetic. This often gives a tigher approximation than (8), because $f[X, z]$ can be significantly smaller than $f'(X)$. Consider, for example, $f(x) = \frac{1}{2}x^2$ and $X = [-1, 3]$. Then $f'(X) = [-1, 3]$ while $f[X, z] = [0, 2]$ for $z = 1$. This is illustrated in Figure 1.

Kolev [15] gives theorems that can be useful in bounding slopes over intervals. For example:

Theorem 1 *Suppose $X \subset \mathbb{R}$ is a finite interval, $z \in X$, and and $f : X \to \mathbb{R}$ is twice continuously differentiable, i.e., $f \in C^2(\mathbb{R})$. If $f''(x) \geq 0 \; \forall \; x \in X$, then $\phi(x) \equiv f[x, z]$ is an increasing function on X (i.e., $\phi'(x) \geq 0$), and if $f''(x) \leq 0 \; \forall \; x \in X$, then $\phi(x)$ is a decreasing function on X (i.e., $\phi'(x) \leq 0$).*

Kolev's proof relies on a property of second-order slopes, but an elementary proof is possible.

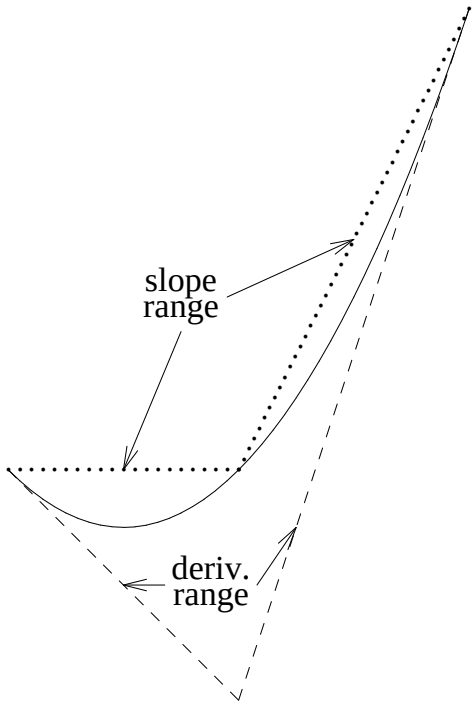

Figure 1: Slopes versus derivative intervals, $f(x) = \frac{1}{2}x^2$ on $[-1, 3]$.

Proof. Since $\phi(x) = \frac{f(x)-f(z)}{x-z}$ for $x \neq z$,

$$\begin{aligned} \phi'(x) &= (x-z)^{-2}\left(f'(x)(x-z) - (f(x)-f(z))\right) \\ &= (x-z)^{-2}\left(f(z) - (f(x)-(x-z)f'(x))\right). \end{aligned}$$

By Taylor's Theorem, $f(z) = f(x) - (x-z)f'(x) + \frac{(x-z)^2}{2}f''(\zeta)$ for some ζ between x and z, so $\phi'(x) = \frac{1}{2}f''(\zeta)$. By continuity, this also holds for $x = z$ (with $\zeta = z$). Since $\zeta \in X$, the result follows. □

Thus on an interval $X = [\underline{x}, \overline{x}]$ where f is convex or concave, $f[X, z]$ is bounded by $f[\underline{x}, z]$ and $f[\overline{x}, z]$.

4 Component-wise Expansions

For general $n > 1$ and $f : \mathbb{R}^n \to \mathbb{R}$, $f[x, z]$ can be defined in various ways. For example, by computing interval enclosures $F_i[X, z]$ of

$$\frac{f(x) - f(z)}{e_i^T(x - z)}$$

for each i, $1 \leq i \leq n$, we would obtain

$$f(X) \subset f(x) + \sum_{i=1}^{n} F_i[X, z](X_i - z_i), \tag{11}$$

when $X = \prod_{i=1}^{n}$ is the Cartesian product of intervals X_i. However, as advocated, e.g., by Hansen [13] and Rump [23], we can often get tighter bounds by expanding one component at a time of X. The idea is still use (11), but with $F_i[X, z]$ enclosing

$$\left\{ \frac{f(x) - f(z)}{e_i^T(x - z)} : x \in X_1 \times X_2 \times ... \times X_i \times \{z_{i+1}\} \times ... \times \{z_n\} \right\}$$

so for $i < j$, $F_i[X, z]$ is bounded over a smaller set than is $F_j[X, z]$, which can result in tighter bounds than when all are bounded over the full X.

With component-wise expansions, permuting the variables may give rise to different over-estimations. In very simple cases, the "best" permutation may be apparent, but in general the only way to find the "best" permutation may be exhaustive search of all $n!$ possibilities. Of course, all are qualitatively the same in the sense of (5), so the choice of variable ordering does not seem like a major concern.

5 Second-Order Slopes

Second-order slopes, first discussed by Zuhe and Wolfe [30], have been defined several ways, including recursive construction rules [30], a divided-difference formula [15] (see (12) below), and as 5- [27] or 7-tuples [26]. I like a divided-difference formula. Let $n = 1$ and $f : \mathbb{R} \to \mathbb{R}$ be continuously differentiable. For $x, z \in \mathbb{R}$ with $x \neq z$,

$$f[x, z, z] = \frac{f[x, z] - f'(z)}{x - z} \tag{12}$$

defines the second-order slope $f[x,z,z]$. If $f \in C^2(\mathbb{R})$, then $\lim_{x\to z} f[x,z,z] = \frac{1}{2}f''(z)$, so we define $f[z,z,z] = \frac{1}{2}f''(z)$. From (12) and (9), we immediately obtain

$$f(x) = f(z) + f'(z)(x-z) + f[x,z,z](x-z)^2. \tag{13}$$

Thus if we have an interval $F[X,z,z] \supset f[X,z,z]$, computing an outer approximation of

$$f(z) + f'(z)(X-z) + F[X,z,z](X-z)^2 \tag{14}$$

with interval arithmetic gives an outer approximation of $f(X)$. This approximation is often tighter than a corresponding outer approximation computed from a first-order slope.

A slightly tighter approximation is sometimes possible, as summarized for convenience in the following theorem.

Theorem 2 *Assume $f : \mathbb{R} \to \mathbb{R}$ is differentiable, and let $z \in \mathbb{R}$ and a compact interval $X \subset \mathbb{R}$ be given. Suppose $[\underline{F}, \overline{F}] \subset \mathbb{R}$ is a compact interval with*

$$f[X,z,z] \subseteq [\underline{F}, \overline{F}]. \tag{15}$$

(a) If $\underline{F} > 0$ and $x^* = z - f'(z)/(2\underline{F}) \in X$, then $f(x) \geq f(z) + f'(z)(x^* - z) + \underline{F}(x^* - z)^2$ for all $x \in X$. Alternatively, (b) if $\overline{F} < 0$ and $x^* = z - f'(z)/(2\overline{F}) \in X$, then $f(x) \leq f(z) + f'(z)(x^* - z) + \overline{F}(x^* - z)^2$ for all $x \in X$.

Proof. From (13) and (15), the quadratic forms

$$\underline{q}(x) = f(z) + f'(z)(x-z) + \underline{F}(x-z)^2$$

and

$$\overline{q}(x) = f(z) + f'(z)(x-z) + \overline{F}(x-z)^2$$

satisfy $\underline{q}(x) \leq f(x) \leq \overline{q}(x)\ \forall\ x \in X$. If (a) holds, then $f(x) \geq \underline{q}(x^*)\ \forall\ x \in X$, and if (b) holds, then $f(x) \leq \overline{q}(x^*)\ \forall\ x \in X$. □

When Theorem 2 applies, we get a tighter bound on one side than evaluation of (14) with interval arithmetic would give.

For bounding second-order slopes over intervals, another result from [15] is useful (and similar to Theorem 1), and again has an elementary proof:

Theorem 3 *Suppose $X \subset \mathbb{R}$ is a finite interval, $z \in X$, and and $f : X \to \mathbb{R}$ is thrice continuously differentiable, i.e., $f \in C^3(\mathbb{R})$. If $f^{(3)}(x) \geq 0\ \forall\ x \in X$, then $\psi(x) \equiv f[x,z,z]$ is an increasing function on X (i.e., $\psi'(x) \geq 0$), and if $f^{(3)}(x) \leq 0\ \forall\ x \in X$, then $\psi(x)$ is a decreasing function on X (i.e., $\phi'(x) \leq 0$).*

Proof. Since

$$\psi(x) = \frac{\frac{f(x)-f(z)}{x-z} - f'(z)}{x-z},$$

$$\begin{aligned} \psi'(x) &= (x-z)^{-2}(f'(x)+f'(z)) - 2(x-z)^{-3}(f(x)-f(z)) \\ &= (x-z)^{-3}\nu(x) \end{aligned}$$

with $\nu(x) \equiv (x-z)(f'(x)+f'(z)) - 2f(x) + 2f(z)$. Now $\nu(z) = 0$, and

$$\nu'(x) = f'(z) - (f'(x) + (z-x)f''(x)).$$

By Taylor's Theorem, $f'(z) = f'(x) + (z-x)f''(x) + \frac{1}{2}(z-x)^2 f^{(3)}(\xi)$ for some ξ between x and z, so $\nu'(x) = \frac{1}{2}(z-x)^2 f^{(3)}(\xi)$, whence $\psi'(x) \geq 0$ if $\inf f^{(3)}(X) \geq 0$ and $\psi'(x) \leq 0$ if $\sup f^{(3)}(X) \leq 0$, and the result follows. □

For $n > 1$, it is possible to define second-order slopes to be $n \times n$ matrices, but it is simpler and perhaps more accurate to proceed as in §4, expanding each operation one dimension at a time and only dealing with one-dimensional second-order slopes, which for the second and higher dimensions must be bounded with interval arithmetic. Specifically, if $X = \prod_{i=1}^{n} [\underline{x}_i, \overline{x}_i]$, when expanding dimension $i > 1$, instead of the point value $f'(z)$ in (12), we must deal with bounds on $\frac{\partial f}{\partial x_i}(Y)$ with

$$Y = [\underline{x}_1, \overline{x}_1] \times ... \times [\underline{x}_{i=1}, \overline{x}_{i-1}] \times \{z_i\} \times ... \times \{z_n\}.$$

6 An Implementation for Expression Graphs

Computing bounds from first- or second-order slopes is analogous to forward automatic differentiation (AD) in that we recur values operation by operation as a computation proceeds. (For much more on AD, see [12] and the references cited therein.) Implementation strategies analogous to those for AD are thus possible, including source-to-source transformations, operator overloading (e.g., in C++), and expression-graph evaluations. For example, in the related area of interval computations, the "XSC languages" [29] use both source-to-source transformations (for Pascal-XSC) and operator overloading (for C-XSC, which now uses templated C++).

Expression-graph evaluations are convenient for some purposes, such as in the AMPL/solver interface library [9], whose evaluations are briefly sketched, e.g., in [8, 10]. See [25] for some related expression-graph computations. An

advantage of expression graphs over templated C++ is that it is easy to do a preliminary graph walk to improve the efficiency of subsequent evaluations. (In principle, this efficiency can also be achieved with templated C++ [28], but I find working with expression graphs much easier than template programming; debugging seems simpler, too.)

interval	$F(X) \supset f(X)$
Taylor 1	$f(z) + F'(X)(X - z)$
slope 1	$f(z) + F[X, z](X - z)$
slope 2	$f(z) + f'(z)(X - z)$ $+F[X, z, z](X - z)^2$
slope 2*	slope 2 plus Theorem 2

Table 1: Bound computations

Starting with source [1] from the AMPL/solver interface library and an "evaluation tester" long intended for addition to [2] when time permits updating [9]), I have written an expression-graph reader that arranges for several kinds of bound computations that are carried out by extended evaluation routines. These include simple interval evaluations, mean-value bounds (8) using interval derivatives, first-order slope bounds (10) that do not do component-wise expansions (§4), and second-order slope bounds (14) with component-wise expansions. The first three bound computations were simply steps to the fourth. Table 1 summarizes these bound computations. In this table, $F(X)$, $F'(X)$, $F[X, z]$, and $F[X, z, z]$ denote outer approximations of $f(X)$, $f'(X)$, $f[X, z]$, and $f[X, z, z]$, respectively, computed with interval arithmetic.

For a function f of n variables whose evaluation involves m operations, computing simple interval bounds also involves $O(m)$ operations. The complexity of the more elaborate bound computations from slopes is $O(nm)$. My implementation exploits sparsity (as determined by the initial tree-walk that sets up the data structures), which reduces the operation count but not the complexity bound. Since this implementation uses loop-free expression graphs, the storage complexity is also $O(nm)$, which of course could be reduced in a more elaborate implementation involving loops or reuse of intermediate storage.

Table 2 reports the widths (3) of the bounds computed as summarized in Table 1 on two examples shown in AMPL [7] format in Figures 2 and 3. Function "Barnes" (Figure 2) is one of the DAKOTA [3] test problems; function "Sn525" (Figure 3) appears in [27].

```
# from $DAKOTA/test/barnes.C
var x1 := 30 in [29,31]; var x2 := 40 in [39,41];
param a{0 .. 20};
minimize f: a[0] + a[1]*x1 + a[2]*x1^2
  + a[3]*x1^3 + a[4]*x1^4 + a[5]*x2 + a[6]*x1*x2
  + a[7]*x1^2*x2 + a[8]*x1^3*x2 + a[9]*x1^4*x2
  + a[10]*x2^2 + a[11]*x2^3 + a[12]*x2^4
  + a[14]*x1^2*x2^2 + a[15]*x1^3*x2^2
  + a[16]*(x1*x2)^3 + a[17]*x1*x2^2
  + a[18]*x1*x2^3
  + a[13]/(x2+1) + a[19]*exp(a[20]*x1*x2);
```

Figure 2: Function "Barnes"

```
# from Schnurr [2008], sec. 5.2

var x{1..3} := 4.125 in [4, 4.25];

minimize f:
  sin(x[1]) + sin((10/3)*x[1])
  + log(x[1]) - 0.84*x[1]
  + 1000*x[1]*x[2]^2*exp(-x[3]^2);
```

Figure 3: Function "Sn525"

For those who wish to try other bound computations with these facilities, it may be helpful to show how the above computations were done. To start with, one needs an AMPL processor; student binaries suffice for $n \leq 300$ and are freely available for various platforms from [4]. Once source for the "evaluation tester", `et`, is available from *netlib*, one might obtain the source and build `et` by clicking "tar" in the line "*lib:* solvers (tar)" of `http://netlib.sandia.gov/ampl`, then (e.g., under Linux) invoke

Method	Barnes	Sn525
interval	162.417	0.7226
Taylor 1	9.350	0.3609
slope 1	6.453	0.3529
slope 2	3.007	0.1140
slope 2*	2.993	0.1003
true	2.330	0.0903

Table 2: Bound widths

```
tar xf netlibfiles.tar
cd solvers
gzip -dN *.gz
sh configurehere
make
cd examples
gzip -dN *.gz
sh configurehere
make et
```

to obtain "`et`". Then, e.g.,

```
ampl -ogbarnes.mod barnes.mod
```

would turn `barnes.mod` containing the text in Figure 2 into file `barnes.nl`, which one could use with "`et`" as follows

```
et
r barnes 6
H 1
F
u
r barnes 7
H 1
F
u
r barnes 8
H 1
u
```

```
r barnes
H 1
F
```

to obtain most of the "Barnes" results shown in Table 2 (all except "slope 2", which requires special compilation to suppress use of Theorem 2). In short, `et` responds to various commands that it summarizes when it sees "?"; the "r" command reads a file in a specified mode (currently an integer between 1 and 9, with 9 the default), arranging for various kinds of evaluations in the process. The portion of "?" output that tells about "r" is

```
r filename mode          load problem; reply with
                         problem number and statistics

  mode values:     1 = linear functions only (f_read)
                   2 = functions and gradients (fg_read)
                   3 = func, grad, and (Lagr.) Hessian (fgh_read)
                   4 = partially sep. func & grad (pfg_read)
                   5 = partially sep. func, grad, Hes (pfgh_read)
                   6 = interval func and grad evaluations
                   7 = mean-value evaluations with interval derivs
                   8 = 1st-order slope evaluations
                   9 = 2nd-order slope evaluations
```

The "H" command specifies half-widths for intervals on independent variables, the "F" command causes bound computations, and the "u" command unloads the current problem.

At this writing, the fourth set of bound computations (involving second-order slopes) is incomplete. Bounding second-order slopes when Theorem 3 does not apply to the whole domain can be tricky. For example, I spent a while looking at the tanh function. It has the nice property that Theorems 1 and 3 apply to portions of its domain. Specifically, $\tanh''(x) > 0$ for $x < 0$, $\tanh''(x) < 0$ for $x > 0$, and there is a point $\alpha \approx 0.6584789484624$ with $\sinh(\alpha)^2 = \frac{1}{2}$ such that $\tanh^{(3)}(x) > 0$ for $x < -\alpha$ or $x > \alpha$ and $\tanh^{(3)}(x) < 0$ for $-\alpha < x < \alpha$. It is possible to break the outer approximation of $\tanh[X, z, z]$ into various cases, some of which involve finding approximate zeros of $\phi'(x)$ with $\phi(x) = f[x, z]$ or $\phi(x) = f[x, z, z]$ and, for rigor (not yet done) penalizing the resulting bounds slightly.

Sometimes there are two or more possible ways to compute a bound, neither of which is always better than the other. Standard practice with interval computations is to compute bounds several ways and intersect them. For example, Rump [23] gives alternate bound computations for some of the elementary arithmetic operations. My implementation uses all the alternative evaluations I am aware of and intersects them.

Initially I used changes of rounding mode [14] as needed to compute rigorous bounds, simply assuming that libray functions (such as log, exp, sin, tanh, etc.) would return correspondingly rounded results. A better approach would be to use FILIB++ [17], or perhaps fdlibm [5] with penalties. (Library routines have sometimes been buggy when it comes to directed roundings. For example, a Linux system I used at Sandia National Labs came with a `libm-2.5.so` that gave $\exp(1) \approx 7.138761293$ rather than 2.718281828 when rounding toward $+\infty$ and gave $\exp(2) \approx 2.718281828$ rather than 7.389056099 when rounding toward $-\infty$. A later version, `libm-2.9.so`, behaves much better.) As various authors point out (see, e.g., §4 of [17]), changing the rounding direction is relatively expensive on some commonly used machines. For example on a system with an Intel Xeon processor (and using the SSE instructions), doing interval multiplications with changes of rounding mode took about 11 times longer than doing everything with rounding toward $-\infty$. The latter is one of several rounding schemes offered by FILIB++, and I hope to modify my implementation to use this rounding exclusively. (For example, if $\bigtriangledown(x)$ and $\bigtriangleup(x)$ denote rounding of x to the largest floating point number $\leq x$ and to the smallest floating-point number $\geq x$, respectively, then we have $\bigtriangleup(x-y) = -\bigtriangledown(y-x)$.) I would set the rounding mode to round towards $-\infty$ at the start of an evaluation routine and restore it to round-nearest just before the routine returned, so as not to cause confusion outside the evaluation routines.

7 Concluding Remarks

This paper discusses use of first- and second-order *slopes* in computing bounds on algebraic expressions. Such bounds are potentially useful in various contexts, such as uncertainty quantification (understanding how inputs known only to lie in specified intervals can affect the values of given expressions), global optimization (a context where many authors have used interval methods), and enforcement of constraints whose violations have dire consequences.

Slopes can also be used to construct convex underestimating and concave overestimating functions and to do constraint propagation [25], i.e., to deduce how constraints imply reduced domains for the variables involved in the constraints, which I have long called "nonlinear presolve". Use of directed roundings in presolve algorithms is discussed, e.g., in [6] and [11].

References

[1] `http://www.netlib.org/ampl/solvers` or `http://netlib.sandia.gov/ampl/solvers`.

[2] `http://www.netlib.org/ampl/solvers/examples` or `http://netlib.sandia.gov/ampl/solvers/examples`.

[3] `http://www.cs.sandia.gov/DAKOTA/`.

[4] `http://www.ampl.com`.

[5] `http://www.netlib.org/fdlibm` or `http://netlib.sandia.gov/fdlibm`.

[6] R. Fourer and D. M. Gay. Experience with a primal presolve algorithm. In W. W. Hager, D. W. Hearn, and P. M. Pardalos, editors, *Large Scale Optimization: State of the Art*, pages 135–154. Kluwer Academic Publishers, 1994.

[7] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press/Brooks/Cole Publishing Co., second edition, 2003.

[8] D. M. Gay. Automatic differentiation of nonlinear AMPL models. In A. Griewank and G. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 61–73. SIAM, 1991.

[9] David M. Gay. Hooking your solver to AMPL. Numerical Analysis Manuscript No. 93-10, AT&T Bell Laboratories, Murray Hill, NJ, 1993, revised 1997.

[10] David M. Gay. More AD of nonlinear AMPL models: Computing Hessian information and exploiting partial separability. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation : Techniques, Applications, and Tools*. SIAM, 1996.

[11] David M. Gay. Symbolic-algebraic computations in a modeling language for mathematical programming. In Götz Alefeld, Jiři Rohn, and Tetsuro Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 99–106. Springer-Verlag, 2001.

[12] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. SIAM, 2008.

[13] Eldon R. Hansen. On solving systems of equations using interval arithmetic. *Math. Computation*, 22(102), 1968.

[14] `http://en.wikipedia.org/wiki/IEEE_754-1985`.

[15] Lubomir V. Kolev. Use of interval slopes for the irrational part of factorable functions. *Reliable Computing*, 3(1), 1997.

[16] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM J. Numer. Anal.*, 22(3), 1985.

[17] Michael Lerch, German Tischler, Jürgen Wolff Von Gudenberg, Werner Hofschuster, and Walter Krämer. Filib++, a fast interval library supporting containment computations. *ACM Trans. Math. Software*, 32(2), 2006.

[18] Ramon E. Moore. Interval arithmetic and automatic error analysis in digital computing. Ph.d. dissertation, Stanford University, 1962.

[19] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[20] Ramon E. Moore. *Methods and Applications of Interval Analysis*. SIAM, 1979.

[21] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[22] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[23] S. M. Rump. Expansion and estimation of the range of nonlinear functions. *Math. of Computation*, 65(216), 1996.

[24] Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. draft article for publication in *Acta Numerica*, 2009.

[25] Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization. *J. Global Optimization*, 33(4), 2005.

[26] Marco Schnurr. Steigungen höherer ordnung zur verifizierten globalen optimierung. Ph.d. dissertation, Universität Karlsruhe, Germany, 2007.

[27] Marco Schnurr. The automatic computation of second-order slope tuples for some nonsmooth functions. *Electronic Transactions on Numerical Analysis*, 30, 2008.

[28] Todd L. Veldhuizen. C++ templates are turing complete. Technical report, Indiana University, Computer Science Dept., 2003.

[29] `http://www.xsc.de/`.

[30] Shen Zuhe and M. A. Wolfe. On interval enclosures using slope arithmetic. *Applied Mathematics and Computation*, 39(1), 1990.